Streamline Python Package Workflows

Automation for Releases, Quality, and More

Will Dean

Who is this talk for?

- Someone trying to get release a Python package with minimal overhead.
- Someone with an idea for a package usable for themselves or their colleagues.
- Demystifying package development.
- Highlighting the realities of maintenance and how to simplify it with automation.

Agenda: The Workflow in Action

- What setting up the GitHub Actions looks like
- What making a release looks like: From Issues to PRs to Releases
- Some pre-commit hooks to look out for
- Tips:
  - When you see a python package you like, check out how they handle their releases and manage their project.

Motivation

- I made a cool Python package and wanted to share and reuse it.
- With a few packages, I noticed maintenance became a challenge.
- I want to focus on coding, not the overhead.

Showcase: `frame-search`

Use `frame-search` to add GitHub search-like functionality to `pandas`, `polars`, and more DataFrame libraries.

Showcase of the package:

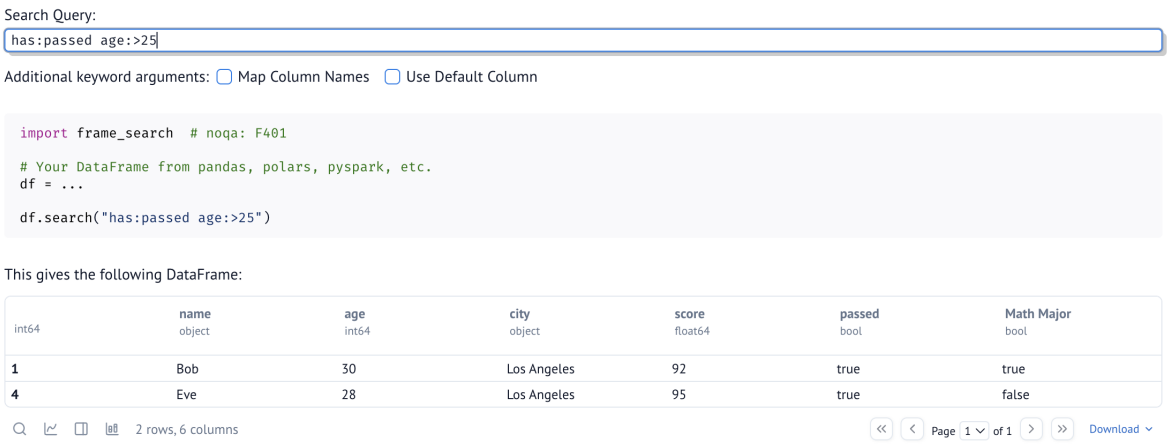
```
import frame_search  # noqa: F401

import polars as pl

data: pl.DataFrame = ...
data.search("column_name:>value another_column:5..10")
```

Showcase: `frame-search` (contd)

This is helpful for filtering DataFrames using string inputs. Say, in a marimo notebook.



The Maintenance Challenge

Why make a package?

- Reusability for yourself and others via `pip install`
- Extend functionality of existing libraries

- Provide different interfaces (e.g., CLI, GUI, API, sugary syntax)

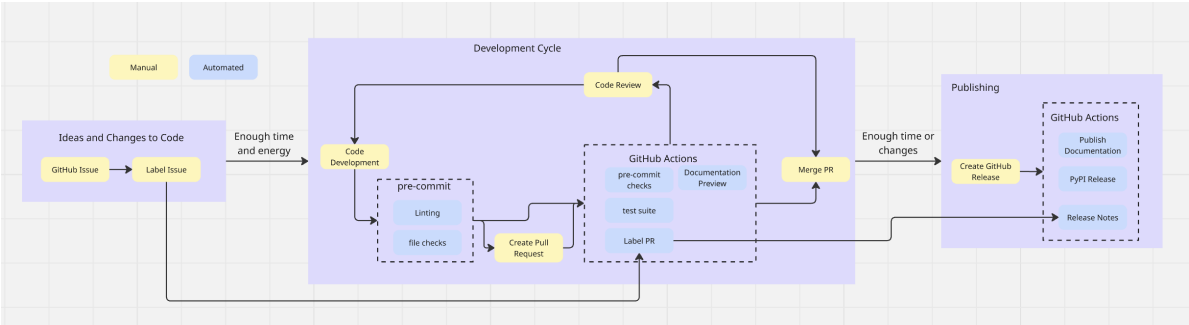
Why not make a package?

- Something already exists
  - Maybe contribute to it instead?
- Overhead of maintenance

The "No-Brainer" Workflow

- **Idea to Code:** Jot down ideas in GitHub Issues with labels.
- **Focus on Coding:** Develop features/fixes.
- **Easy Release:** Update version and make a GitHub Release.
- **Goodies for Free:** Automated release notes, CI/CD, PyPI publish, docs!

Overview of the Automated Workflow



Background on Automation Tools

`pre-commit` (local)

- Run before each commit
- Defined in `.pre-commit-config.yaml`
- Find hooks on pre-commit.com/hooks

`GitHub Actions` (remote)

- Triggered on GitHub defined events (push, PR, schedule, manual, release trigger, comment)
- Defined in `.github/workflows/*.yml`
- Find Actions on GitHub Marketplace

Which Tool for Which Job?

- Keep the local dev environment snappy with `pre-commit` hooks.
  - Linting, formatting, type-checking, doc checks.
- Offload other tasks to `GitHub Actions` CI/CD.
  - Long running tasks like tests, type checking, building docs
  - Publishing to PyPI, deploying documentation
  - Rely on different triggers like issue updates, PR merges, releases.

Deep Dive #1: Publishing

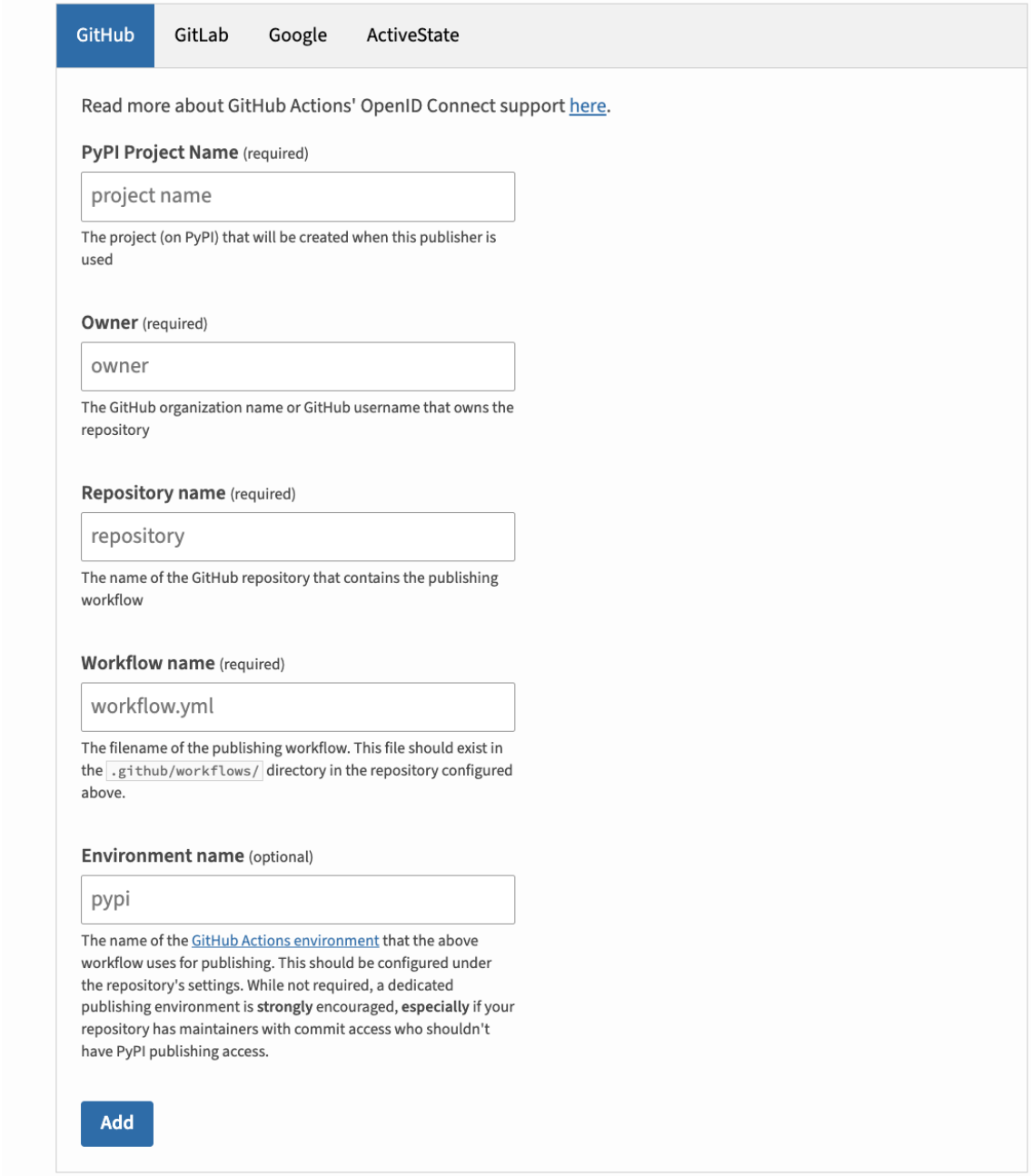
```yaml
# .github/workflows/publish.yml
---
name: Python package
on:
  push:
    tags:
      - "v*.*.*"
jobs:
  build:
```

```yaml
    runs-on: ubuntu-latest
    permissions:
      id-token: write
    steps:
      - uses: actions/checkout@v5
      - uses: astral-sh/setup-uv@v7
      - run: uv build
      - run: uv publish --trusted-publishing always
```

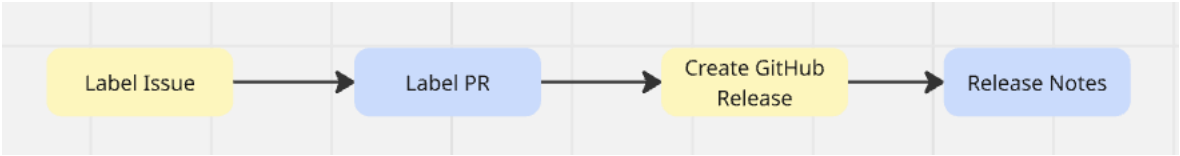
Deep Dive #1: Publishing (contd)

- Register Action as a trusted publisher on pypi.org
- pypi.org -> manage -> account -> publishing

- Will not require username/password i.e. `uv publish` just works!


GitHub
GitLab
Google
ActiveState
Read more about GitHub Actions' OpenID Connect support here.
PyPI Project Name (required)
project name
The project (on PyPI) that will be created when this publisher is used
Owner (required)
owner
The GitHub organization name or GitHub username that owns the repository
Repository name (required)
repository
The name of the GitHub repository that contains the publishing workflow
Workflow name (required)
workflow.yml
The filename of the publishing workflow. This file should exist in the .github/workflows/ directory in the repository configured above.
Environment name (optional)
pypi
The name of the GitHub Actions environment that the above workflow uses for publishing. This should be configured under the repository's settings. While not required, a dedicated publishing environment is strongly encouraged, especially if your repository has maintainers with commit access who shouldn't have PyPI publishing access.
Add

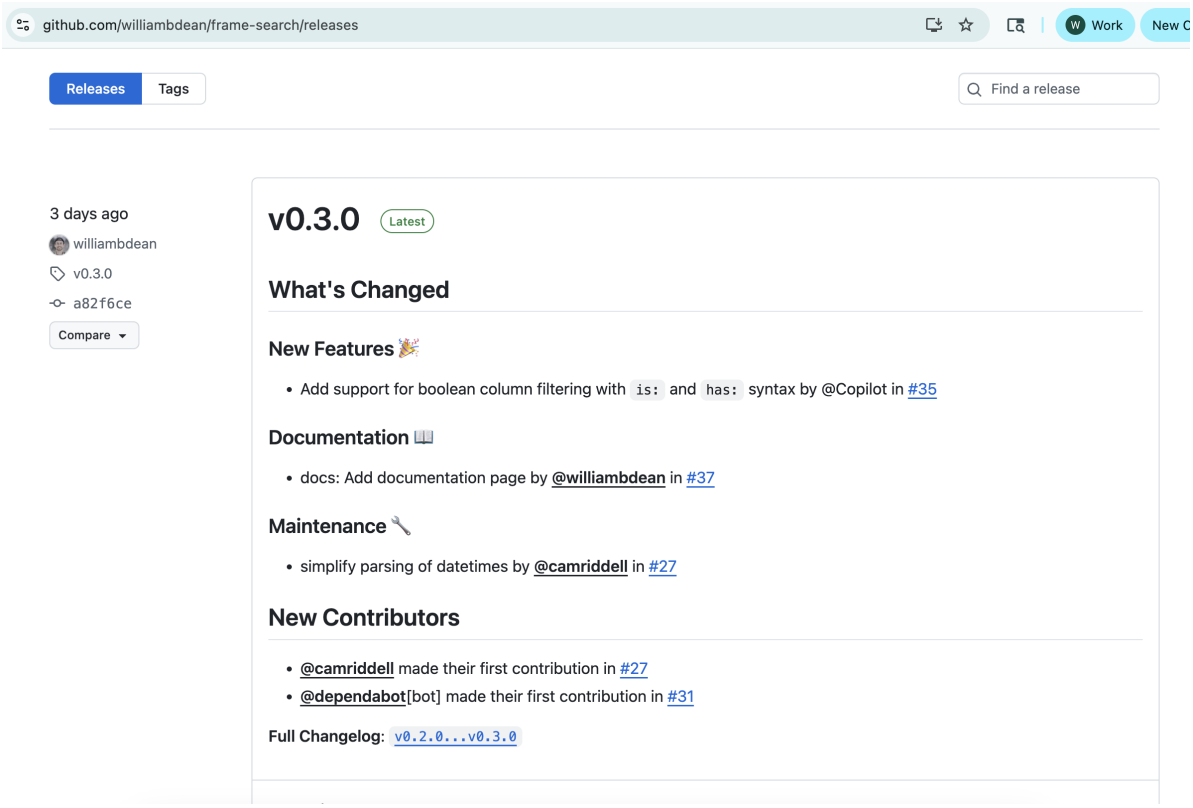
Deep Dive #2: Automated Release Notes



Manual touchpoints to the process:

- Label the issue while creating it (e.g., `enhancement`, `bug`).
- Creating the GitHub Release:
  - Draft a Release > Select tag > Generate release notes
  - `gh release create <tag> --generate-notes`

Deep Dive #2: Automated Release Notes (contd)



Deep Dive #2: Automated Release Notes (contd)

GitHub supported automated release notes generation.

```yaml
# .github/release.yml
changelog:
  exclude:
    labels: ["no releasenotes"]
  categories:
    - title: New Features
      labels: ["enhancement"]
    - title: Bugfixes
      labels: ["bug"]
    - title: Documentation
      labels: ["docs"]
    - title: Maintenance
      labels: ["*"]
```

Deep Dive #2: Automated Release Notes (contd)

```yaml
# .github/workflows/sync-closing-labels.yml
---
name: Sync Closing Labels
on:
- pull_request_target

jobs:
  sync:
    permissions:
      pull-requests: write
    runs-on: ubuntu-latest
    steps:
    - name: Sync labels for a pull request with its linked issues
      uses: williambdean/closing-labels@v0.0.6
      env:
        GH_TOKEN: ${{ github.token }}
```

Deep Dive #3: Pre-commit Hooks

Don't forget `pre-commit install`!

```yaml
# .pre-commit-config.yaml
---
repos:
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.14.2
    hooks:
      - id: ruff-check
        types_or: [python, pyi]
        args: ["--fix", "--output-format=full"]
      - id: ruff-format
        types_or: [python, pyi]

  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v6.0.0
    hooks:
      - id: debug-statements
      - id: trailing-whitespace
      - id: end-of-file-fixer
      - id: check-toml
      - id: check-yaml
      - id: check-added-large-files
        exclude: ^docs/
```

Deep Dive #4: CI/CD for Testing

```yaml
# .github/workflows/tests.yml
---
name: Tests

# Trigger only of relevant paths change
on:
  push:
    paths:
      - "src/**"
```

```yaml
      - "pyproject.toml"
      - "tests/**"
  pull_request:
    paths:
      - "src/**"
      - "pyproject.toml"
      - "tests/**"

jobs:
  test:
    name: python
    runs-on: ubuntu-latest

    # Run on multiple Python versions in parallel
    strategy:
      matrix:
        python-version:
          - "3.9"
          - "3.10"
          - "3.11"
          - "3.12"
          - "3.13"
    env:
      UV_PYTHON: ${{ matrix.python-version }}

    steps:
      - uses: actions/checkout@v5

      - name: Install uv
        uses: astral-sh/setup-uv@v7

      - name: Install the package with dev dependencies
        run: uv sync --locked --all-extras --dev

      - name: Run test suite
        run: uv run pytest tests/
```

Summary

Just a few yaml files to set up a robust workflow!

```
.github
├── dependabot.yml
├── release.yml
└── workflows
    ├── publish.yml
    ├── sync-closing-labels.yml
    └── tests.yml
.pre-commit-config.yaml
```

Key Takeaways & Tips

- **Start with pain points; it's progressive.** Linting is crucial (code quality, doc errors, style consistency).
- **Automation pays off:** Low maintenance, reusable workflows. Automate high-pain tasks first.
- **Contribute wisely:** Gauge project, respect maintainers. Create your own for extensions.
- **Embrace automation:** Let tools handle complexity.
- **Learn & adapt:** Observe other projects' release strategies (e.g., `narwhals` cat emojis!).

Draft Slides

Q&A / Resources

- Your GitHub Profile
- frame-search repository
- Relevant PEPs (518, 621)

- pre-commit hooks website
- uv documentation
- narwhals documentation
- Appendix

Deep Dive #3: Pre-commit Hooks

- Crucial for local quality checks (linting, formatting, type-checking).
- **Fast feedback:** Aim for 5-10 second runtime. Not for long test suites (use CI/CD).
- Run in CI/CD too (catches local bypasses).
- Examples: Docs checks, Linting (Ruff), Type checking (MyPy, `ty`), Code trimming.
- **Custom Hooks:** Run local scripts for tailored automation. Many available on `pre-commit.com`.

Deep Dive: Labels & Automated Release Notes

- Configured via `.github` file (defines categories & order).
- **Action:** `williambdean/closing-labels` transfers labels from issues to PRs.
  - Label issues when jotting down ideas; no need to label PRs.
- **Prioritize:** Breaking > New Features > Bugs > Maintenance.
- Multiple labels: PR appears under first matching category.
- Use `no-releasenotes` for trivial changes.
- Edit notes post-generation (GitHub UI) for enrichment.

Deep Dive: CI/CD for Testing (and using test PyPI)

- CI/CD: Testing, lock files, docs, PyPI publish, doc deployment.
- Failures: Rerun (fluke) or fix (error).
- **Minimal overhead:** PyPI steps ~2-5 min. (Test suites are usually longer).
- **Pro-tip:** Use **test PyPI** for pre-release validation (e.g., on PRs).
  - Example: `pymc-marketing` workflow installs from test PyPI and checks import/version.

Deep Dive: The Release Process

- Manual version bump in a PR (e.g., `uv version --bump major`).
- GitHub Release (UI) triggers:
  - Automated PyPI publishing.
  - Automated documentation deployment.
- **Fast turnaround:** 2-5 min from merge to PyPI live.

Tools Used

- GitHub Labels
- GitHub CLI
- GitHub Actions
  - actions/labeler
  - williambdean/closing-labels
- pre-commit
- ruff for formatting and linting
- uv for package management and publishing (with `uvx` for trying out tools)
- pytest
- narwhals
- pandas and polars API extensions
- mkdocs or marimo for documentation

Key Takeaways & Tips

- **Start with pain points; it's progressive.** Linting is crucial (code quality, doc errors, style consistency).
- **Automation pays off:** Low maintenance, reusable workflows. Automate high-pain tasks first.
- **Contribute wisely:** Gauge project, respect maintainers. Create your own for extensions.
- **Embrace automation:** Let tools handle complexity.
- **Learn & adapt:** Observe other projects' release strategies (e.g., `narwhals` cat emojis!).

Q&A / Resources

- Your GitHub Profile
- frame-search repository
- Relevant PEPs (518, 621)
- pre-commit hooks website
- uv documentation
- narwhals documentation
- Appendix

Appendix

Alternative to PyPI: Install from GitHub

- You can install packages directly from GitHub. This allows much of the workflow (CI/CD, etc.) to still be in place without the PyPI release step.

`frame-search` Pain Point

- Handling advanced, nested logic when parsing search strings.
- Balancing regex with grammar-type libraries while keeping it maintainable within Python, avoiding obscure new syntaxes.

`frame-search` Custom Operators

- Current customization is primarily mapping column names to alternative, user-friendly keys.
- Cannot define new operators like `is_weekend:date` unless it's an existing column.
- Future idea: extending `is:` operator for Boolean columns.
- Offers flexibility for filtering on dates, numerical values, ranges.

`frame-search` Performance

- Performance depends on the underlying `narwhals` backend (Polars, Pandas, etc.).
- Users typically know their backend; with Polars LazyFrames, explicit `collect()` is needed.
- Primary use case is interactive scenarios (e.g., Marimo) and as an alternative to other filtering logic.

Biggest Maintenance Headache

- Risk of version updates, dependency management (tight/loose constraints).
- Yearly Python updates.
- Mitigation: Keep packages lean, follow projects/maintainers.

Biggest Brain-Drain Eliminated

- The *personalization* and *predictability* of the workflow.
- Automating tasks like the GitHub release trigger or issue labeling makes the process easy and predictable.

Most Immediate ROI

- Automating **documentation and PyPI release**.
- Releases are infrequent, making it hard to remember steps; automation removes cognitive load and reduces errors.

Pre-commit vs. CI/CD

- **Rule of thumb:** If a check takes > ~30 seconds, push to CI/CD. Aim for 5-10 seconds for pre-commit.
- Pre-commit for snappy local feedback; CI/CD for longer-running tasks.
- Optimize CI/CD with `pytest-split`, caching, and fast installers like `uv`.

Pre-commit Hooks to Avoid

- Potentially slow type checking hooks like `MyPy` if not implemented from the start.
- Emphasizes running `pre-commit run --all-files` and running pre-commit in CI/CD for consistency.

`ty` for Type Checking

- Experimented with `uvx` but not in production due to perceived (unidentified) instability.
- Aware of its potential and that the team seeks feedback.

Customizing Release Notes Format

- Primarily uses GitHub's auto-generated notes; not sure how to add custom headers/footers.
- Learns by observing other projects' templates and trying to recreate them.

Alternative Tools

- **CI/CD:** GitHub Actions (built-in preference); aware of GitHub Apps/marketplace actions.
- **Data manipulation:** `Ibis` (supports many backends), but `narwhals` also supports `Ibis`.

Inspired Projects

- PyMC, PyMC-Marketing.
- `narwhals` project (clean automation, cat emojis in PRs).
- General approach: Observe consistent quality/clean processes in other projects.

Single Most Important Advice

- **Start with pain points; it's progressive.**
- Prioritize linting (code quality, doc error catching, style consistency).
- Automate high-pain, high-uncertainty tasks like publishing first.